

# A Low-Complexity Improved Successive Cancellation Decoder for Polar Codes

Orion Afisiadis, Alexios Balatsoukas-Stimming, and Andreas Burg

Telecommunications Circuits Laboratory, École Polytechnique Fédérale de Lausanne, Switzerland.

**Abstract**—Under successive cancellation (SC) decoding, polar codes are inferior to other codes of similar blocklength in terms of frame error rate. While more sophisticated decoding algorithms such as list- or stack-decoding partially mitigate this performance loss, they suffer from an increase in complexity. In this paper, we describe a new flavor of the SC decoder, called the *SC flip* decoder. Our algorithm preserves the low memory requirements of the basic SC decoder and adjusts the required decoding effort to the signal quality. In the waterfall region, its average computational complexity is almost as low as that of the SC decoder.

## I. INTRODUCTION

Polar codes [1] are particularly attractive from a theoretical point of view because they are the first codes that are both highly structured and provably optimal for a wide range of applications (in the sense of optimality that pertains to each application). Moreover, they can be decoded using an elegant, albeit suboptimal, successive cancellation (SC) algorithm, which has computational complexity  $O(N \log N)$  [1], where  $N = 2^n$ ,  $n \in \mathbb{Z}$ , is the blocklength of the code, and memory complexity  $O(N)$  [2]. Even though the SC decoder is suboptimal, it is sufficient to prove that polar codes are capacity achieving in the limit of infinite blocklength.

Unfortunately, the error correcting performance of SC decoding at finite blocklengths is not as good as that of other modern codes, such as LDPC codes. To improve the finite blocklength performance, more sophisticated algorithms, such as *SC list* decoding [3] and *SC stack* decoding [4], were introduced recently. These algorithms use SC as the underlying decoder, but improve its performance by exploring multiple paths on a decision tree simultaneously, with each path resulting in one candidate codeword. The computational and memory complexities of SC list decoding are  $O(LN \log N)$  and  $O(LN)$ , respectively, where  $L$  is the *list size* parameter, whereas the computational and memory complexities of SC stack decoding are  $O(DN \log N)$  and  $O(DN)$ , respectively, where  $D$  is the *stack depth* parameter.

Since an exhaustive search through all paths is prohibitively complex, choosing a suitable strategy for pruning unlikely paths is an important ingredient for low-complexity tree search algorithms. To this end, in [4], some path pruning-based methods were proposed in order to reduce the computational complexity of both SC stack and SC list decoding. An alternative approach to reduce the computational complexity of SC list decoding was taken in [5], [6], where decoding starts with list size 1, and the list size is increased only when decoding fails (failures are detected using a CRC), up to the maximum list size  $L$ . Moreover, in [7] SC list decoding is employed only

for the least reliable bits of the polar code, thus also reducing the computational complexity. However, in [7]  $L$  distinct paths are still followed in parallel.

Unfortunately, when implementing any decoder in hardware, one always has to provision for the worst case in terms of hardware resources. For the reduced-complexity SC list decoders in [4]–[7] and the reduced-complexity SC stack decoder in [4] this means that  $O(LN)$  and  $O(DN)$  memory needs to be instantiated, respectively. Moreover, the reduced-complexity list SC and stack SC algorithms also have a significantly higher computational complexity than that of the original SC algorithm.

**Contribution:** In this paper, we describe a new SC-based decoding algorithm, called *SC flip*, which retains the  $O(N)$  memory complexity of the original SC algorithm and has an average computational complexity that is practically  $O(N \log N)$  at high SNR, while still providing a significant gain in terms of error correcting performance.

## II. POLAR CODES AND SUCCESSIVE CANCELLATION DECODING

### A. Construction of Polar Codes

Let  $W$  denote a binary input memoryless channel with input  $u \in \{0, 1\}$ , output  $y \in \mathcal{Y}$ , and transition probabilities  $W(y|u)$ . A polar code is constructed by recursively applying a  $2 \times 2$  *channel combining* transformation on  $2^n$  independent copies of  $W$ , followed by a *channel splitting* step [1]. This results in a set of  $N = 2^n$  synthetic channels, denoted by  $W_n^{(i)}(y_1^N, u_1^{i-1}|u_i)$ ,  $i = 1, \dots, N$ . Let  $Z_i \triangleq Z(W_n^{(i)}(Y_1^N, U_1^{i-1}|U_i))$ ,  $i = 1, \dots, N$ , where  $Z(W)$  is the Bhattacharyya parameter of  $W$ , which can be calculated using various methods (cf. [1], [8], [9]). The construction of a polar code of rate  $R \triangleq \frac{k}{N}$ ,  $0 < k < N$ , is completed by choosing the  $k$  best synthetic channels (i.e., the synthetic channels with the lowest  $Z_i$ ) as *non-frozen* channels which carry information bits, while *freezing* the input of the remaining channels to some values  $u_i$  that are known both to the transmitter and to the receiver. The set of frozen channel indices is denoted by  $\mathcal{A}^c$  and the set of non-frozen channel indices is denoted by  $\mathcal{A}$ . The encoder generates a vector  $u_1^N$  by setting  $u_{\mathcal{A}^c}$  equal to the known frozen values, while choosing  $u_{\mathcal{A}}$  freely. A codeword is obtained as  $x_1^N = u_1^N G_N$ , where  $G_N$  is the generator matrix [1].

### B. Successive Cancellation Decoding

The SC decoding algorithm [1] starts by computing an estimate of  $u_1$ , denoted by  $\hat{u}_1$ , based only on the received

values  $y_1^N$ . Subsequently,  $u_2$  is estimated using  $(y_1^N, \hat{u}_1)$ , etc. Since  $u_i$ ,  $i \in \mathcal{A}^c$  are known to the receiver, the real task of SC decoding is to estimate  $u_i$ ,  $i \in \mathcal{A}$ . Let the log-likelihood ratio (LLR) for  $W_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i)$  be defined as

$$L_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i) \triangleq \log \frac{W_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i = 0)}{W_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i = 1)}. \quad (1)$$

Decisions are taken according to

$$\hat{u}_i = \begin{cases} 0, & L_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i) \geq 0 \text{ and } i \in \mathcal{A}, \\ 1, & L_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i) < 0 \text{ and } i \in \mathcal{A}, \\ u_i, & i \in \mathcal{A}^c. \end{cases} \quad (2)$$

The decision LLRs  $L_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i)$  can be calculated efficiently through a computation graph which contains two types of nodes, namely  $f$  nodes and  $g$  nodes. An example of this graph for  $N = 8$  is given in Fig. 1. Both types of nodes have two input LLRs, denoted by  $L_1$  and  $L_2$ , and one output LLR, denoted by  $L$ . The  $g$  nodes have an additional input called the *partial sum*, denoted by  $u$ . The partial sums form the *decision feedback* part of the SC decoder. The min-sum update rules [2] for the two types of nodes are

$$f(L_1, L_2) = \text{sign}(L_1)\text{sign}(L_2) \min(|L_1|, |L_2|), \quad (3)$$

$$g(L_1, L_2, u) = (-1)^u L_1 + L_2. \quad (4)$$

The partial sums at stage  $(s-1)$  can be calculated from the partial sums at stage  $s$ ,  $s \in \{1, \dots, n\}$ , as

$$u_{s-1}^{(2i-1-[(i-1) \bmod 2^{s-1}])} = u_s^{(2i-1)} \oplus u_s^{(2i)}, \quad (5)$$

$$u_{s-1}^{(2^{s-1}+2i-1-[(i-1) \bmod 2^{s-1}])} = u_s^{(2i)}, \quad (6)$$

where

$$u_n^{(i)} \triangleq \hat{u}_i, \quad \forall i \in \{1, \dots, N\}. \quad (7)$$

The computation graph contains  $N \log(N+1)$  nodes and each node only needs to be activated once. Thus, the computational complexity of SC decoding is  $O(N \log N)$ . A straightforward implementation of the computation graph in Fig. 1 requires  $O(N \log N)$  memory positions. However, by cleverly re-using memory locations, it is possible to reduce the memory complexity to  $O(N)$  [2].

### III. ERROR PROPAGATION IN SC DECODING

In SC decoding, erroneous bit decisions can be caused by channel noise *or* by error propagation due to previous erroneous bit decisions. The first erroneous decision is always caused by the channel noise since there are no previous errors, so error propagation does not affect the frame error rate of polar codes, but only the bit error rate.

#### A. Effect of Error Propagation

The erroneous decisions due to error propagation are caused by erroneous decision feedback, which in turns leads to erroneous partial sums. Erroneous partial sums can corrupt the output LLR values at all stages, including, most importantly, the decision LLRs at level  $n$ .

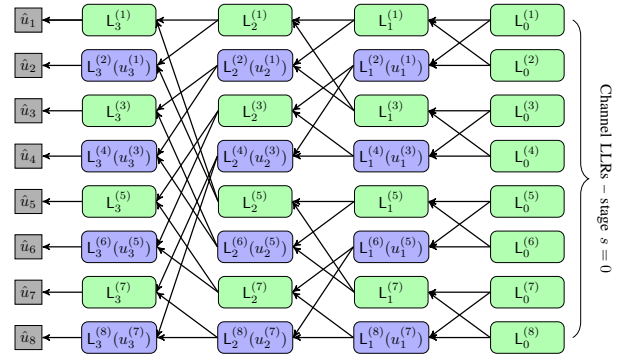


Fig. 1: The computation graph of the SC decoder for  $N = 8$ . The  $f$  nodes are green and  $g$  nodes are blue and in the parentheses are the partial sums that are used by each  $g$  node.

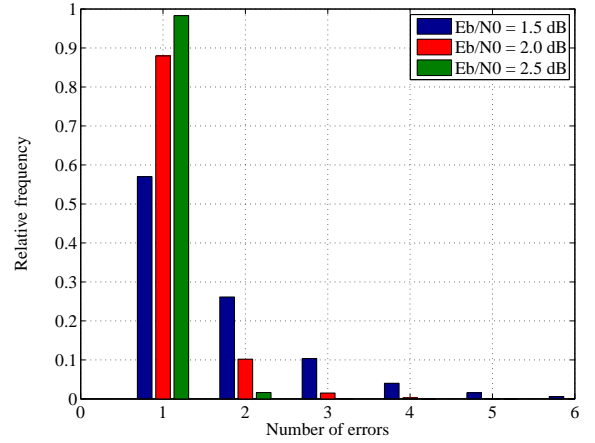


Fig. 2: Histogram showing the relative frequency of the number of errors caused by the channel for a polar code with  $N = 1024$  and  $R = 0.5$  for three different SNR values.

For example, assume that, for the polar code in Fig. 1, the frozen set is  $\mathcal{A}^c = \{1, 2, 5, 6\}$  and the information set is  $\mathcal{A} = \{3, 4, 7, 8\}$ . Moreover, assume that the all-zero codeword was transmitted and that  $\hat{u}_3$  was erroneously decoded as  $\hat{u}_3 = 1$  due to channel noise. Now suppose that the two LLRs that are used to calculate the next decision LLR (i.e.,  $L_3^{(4)}$ ), namely,  $L_2^{(2)}$  and  $L_2^{(6)}$ , are both positive and  $L_2^{(2)} > L_2^{(6)}$ . By applying the  $g$  node update rule with  $u = u_3^{(3)} = \hat{u}_3 = 1$ , the resulting decision LLR  $L_3^{(4)} = L_2^{(6)} - L_2^{(2)}$  has a negative value which leads to a second erroneous decision, while with the correct partial sum  $u = 0$  the decision would have been correct.

#### B. Significance of Error Propagation

The foregoing analysis of the effects of error propagation insinuates the following question: Given that we had an erroneously decoded codeword with many erroneous bits, how many of these bits were actually wrong because of channel noise rather than due to previous erroneous decisions? In order to answer to this question, we employ an oracle-assisted SC decoder. Each time an error occurs at the decision level, the oracle corrects it instantaneously without allowing it to

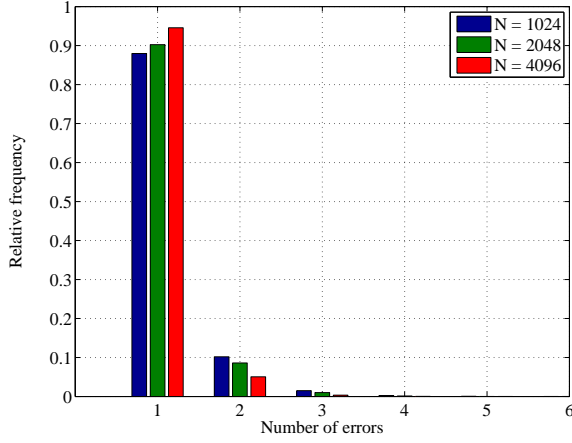


Fig. 3: Histogram showing the relative frequency of the number of errors actually caused by the channel for  $E_b/N_0 = 2.00$  and three different codeword lengths,  $N = 1024, 2048, 4096$ .

affect any future bit decisions. Moreover, the oracle-assisted SC decoder counts the number of times it had to correct an erroneous decision.

In Fig. 2 we plot a histogram of the number of errors caused by channel noise (given that there was at least one error) for three different  $E_b/N_0$  values for a polar code with  $N = 1024$  and  $R = 0.5$  over an AWGN channel. We observe that most frequently the channel introduces only one error and that this behavior becomes even more prominent for increasing  $E_b/N_0$  values. In Fig. 3 we plot a histogram of the number of errors caused by channel noise for polar codes with three different blocklengths and  $R = 0.5$  over an AWGN channel at  $E_b/N_0 = 2$  dB. We observe that, the relative frequency of the single error event increases with increasing blocklengths. This happens because, as  $N$  gets larger, the synthetic channels  $W_n^{(i)}(y_1^N, u_1^{i-1}|u_i)$  become more polarized, meaning that all information channels in  $\mathcal{A}$  become better.

### C. Oracle-Assisted SC Decoder

From the discussion in the previous section, it is clear that, by identifying the position of the first erroneous bit decision and inverting that decision, the performance of the SC decoder could be improved significantly. In order to examine the potential benefits of correcting a single error we employ a second oracle-assisted SC decoder, which is only allowed to intervene *once* in the decoding process in order to correct the first erroneous bit decision.

In Fig. 4 we compare the performance of the SC decoder with that of the oracle-assisted SC decoder for a polar code of three blocklengths and  $R = 0.5$  over an AWGN channel. We observe that correcting a single erroneous bit decision significantly improves the performance of the SC decoder.

## IV. SC FLIP DECODING

The goal of SC flip decoding is to identify the first error that occurs during SC decoding without the aid of an oracle.

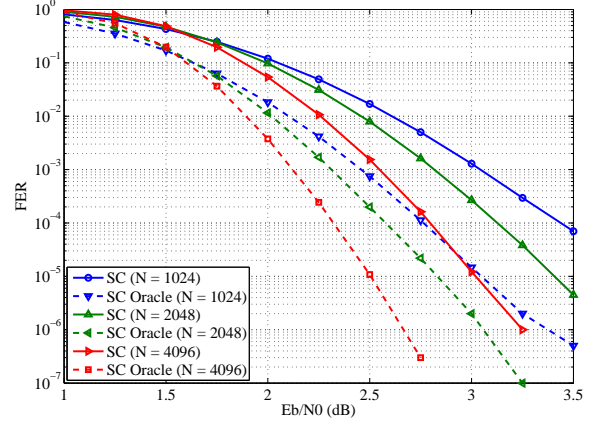


Fig. 4: Performance of oracle-assisted SC decoder compared to the SC decoder for  $N = 1024, 2048, 4096$  and  $R = 0.5$ .

### A. SC Flip Decoding Algorithm

Assume that we are given a polar code of rate  $\tilde{R} = \frac{k}{N}$  with a set of information bits  $\tilde{\mathcal{A}}$ . We use an  $r$ -bit CRC that tells us, with high probability, whether the codeword estimate  $\hat{u}_1^N$  given by the SC decoder is a valid codeword or not. In order to incorporate the CRC, the rate of the polar code is increased to  $R = \tilde{R} + \frac{r}{N} = \frac{k+r}{N}$ , so that the effective information rate remains unaltered. Equivalently, the set of information bits  $\tilde{\mathcal{A}}$  is extended with the  $r$  most reliable channel indices in  $\tilde{\mathcal{A}}^c$ , denoted by  $\tilde{\mathcal{A}}_{r-\max}^c$ . Thus,  $\mathcal{A} = \tilde{\mathcal{A}} \cup \tilde{\mathcal{A}}_{r-\max}^c$ .

The SC flip decoder starts by performing standard SC decoding in order to produce a first estimated codeword  $\hat{u}_1^N$ . If  $\hat{u}_1^N$  passes the CRC, then decoding is completed. If the CRC fails, the SC flip algorithm is given  $T$  additional attempts to identify the first error that occurred in the codeword. To this end, let  $\mathcal{U}$  denote the set of the  $T$  least reliable decisions, i.e., the set containing the indices  $i \in \mathcal{A}$  corresponding to the  $T$  smallest  $|L_n^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i)|$  values. After the set  $\mathcal{U}$  has been constructed, SC decoding is restarted for a total of no more than  $T$  additional attempts. In each attempt, a single  $\hat{u}_k, k \in \mathcal{U}$ , is flipped with respect to the initial decision of the SC algorithm. The algorithm terminates when a valid codeword has been found or when all  $T$  additional attempts have failed. Note that, for  $T = 0$ , SC flip decoding is equivalent to SC decoding.

The SC flip algorithm is formalized in the  $\text{SCFLIP}(y_1^N, \mathcal{A}, k)$  function in Fig. 5. The  $\text{SC}(y_1^N, \mathcal{A}, k)$  function performs SC decoding based on the channel output  $y_1^N$  and the set of non-frozen bits  $\mathcal{A}$  with a slight twist: when  $k > 0$ , the codeword bit  $u_k$  is decoded by flipping the value obtained from the decoding rule (2).

Note that SC flip decoding is similar to chase decoding for polar codes [10]. The main differences are that SC flip decoding only considers error patterns containing a single error and that these error patterns are not generated offline using the a-priori reliabilities of the synthetic channels  $W_n^{(i)}(y_1^N, u_1^{i-1}|u_i)$ , but online using the decision LLRs  $L_n^{(i)}$ , which reflect the actual reliabilities of the bit decisions for each transmitted

```

1: function SCFLIP( $y_1^N, \mathcal{A}, T$ )
2:  $(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow \text{SC}(y_1^N, \mathcal{A}, 0)$ ;
3: if  $T > 1$  and  $\text{CRC}(\hat{u}_1^N) = \text{failure}$  then
4:    $\mathcal{U} \leftarrow i \in \mathcal{A}$  of  $T$  smallest  $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|$ ;
5:   for  $j \leftarrow 1$  to  $T$  do
6:      $k \leftarrow \mathcal{U}(j)$ ;
7:      $\hat{u}_1^N \leftarrow \text{SC}(y_1^N, \mathcal{A}, k)$ ;
8:     if  $\text{CRC}(\hat{u}_1^N) = \text{success}$  then
9:       break;
10:    end if
11:  end for
12: end if
13: return  $\hat{u}_1^N$ ;

```

Fig. 5: SC flip decoding with maximum trials  $T$ .

codeword and channel noise realization.

### B. Complexity of SC Flip Decoding

In this section, we derive the worst-case and average-case computational complexities of the SC flip algorithm, as well as its memory complexity.

**Proposition 1.** *The worst-case computational complexity of the SCFLIP algorithm defined in Fig. 5 is  $O(TN \log N)$ .*

*Proof:* SC decoding in line 2 has complexity  $O(N \log N)$  and the computation of the CRC in line 3 has complexity  $O(N)$ . Moreover, the sorting step in line 4 can be implemented with complexity  $O(N \log N)$  (e.g., using merge sort). Finally, the operations in the loop (lines 5–11) have complexity  $O(N \log N)$  and the loop runs  $T$  times in the worst case. Thus, the overall worst-case complexity is  $O(TN \log N)$ . ■

Proposition 1 shows that, in the worst case, the complexity of our algorithm increases linearly with the parameter  $T$ , meaning that its complexity scaling is no better than that of SC list decoding. However, if we consider the *average* complexity, then the situation is much more favorable, as the following result shows.

**Proposition 2.** *Let  $P_e(R, \text{SNR})$  denote the frame error rate of a polar code of rate  $R$  at the given SNR point. Then, the average-case computational complexity of the SCFLIP algorithm defined in Fig. 5 is  $O(N \log N(1 + T \cdot P_e(R, \text{SNR})))$ , where  $R = \frac{k+r}{N}$ .*

*Proof:* It suffices to observe that the loop in lines 5–11 runs only if SC decoding fails and the CRC detects the failure, which happens with probability at most  $P_e(R, \text{SNR})$ . ■

As the SNR increases, the FER drops asymptotically to zero. Thus, for high SNR the average computational complexity of SC flip decoding converges to the computational complexity of SC decoding. In other words, SC flip exhibits an *energy-proportional* behavior where more energy is spent when the problem is difficult (i.e., at low SNR) and less energy is spent when the problem is easy (i.e., at high SNR).

**Proposition 3.** *The SCFLIP algorithm defined in Fig. 5 requires  $O(N)$  memory positions.*

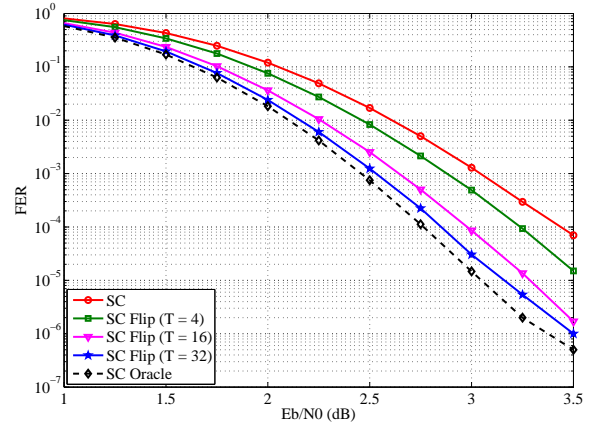


Fig. 6: Frame error rate of SC decoding, SC flip decoding with  $T = 4, 16, 32$  and the oracle-assisted SC decoder for a polar code of length  $N = 1024$  and  $R = 0.5$ .

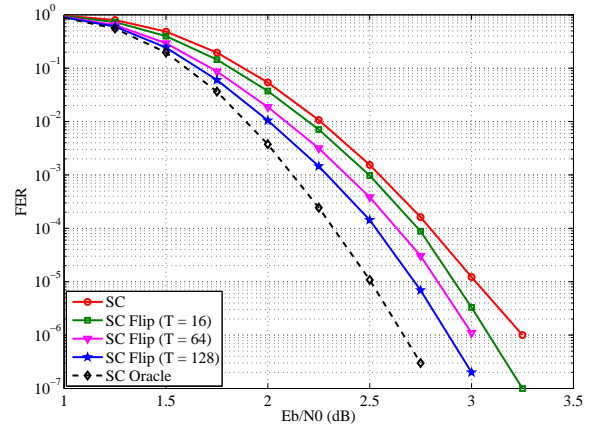


Fig. 7: Frame error rate of SC decoding, SC flip decoding with  $T = 4, 16, 32$  and the oracle-assisted SC decoder for a polar code of length  $N = 4096$  and  $R = 0.5$ .

*Proof:* SC decoding in line 2 requires  $O(N)$  memory positions. The storage of the CRC calculated in lines 3 and 8 requires exactly  $C$  memory positions, where  $C \leq N$ . The sorting step in line 4 can be implemented with  $O(N)$  memory positions (e.g., using merge sort), while storing the  $T$  smallest values requires exactly  $T$  memory positions, where  $T \leq N$ . Moreover, the SC decoding performed in line 7 can re-use the memory positions of the SC decoding in line 2, so no additional memory is required. Thus, the overall memory scaling behavior is  $O(N)$ . ■

### C. Error Correcting Performance of SC Flip Decoding

In Fig. 6 we compare the performance of the SC flip decoder with  $T = 4, 16, 32$ , and a 16-bit CRC with the SC decoder and the oracle-assisted SC decoder described in Section III-C. Note that the oracle-assisted decoder characterizes a performance bound for the SC flip decoder. We observe that SC flip decoding with  $T = 4$  already leads to a gain of one order of magnitude in terms of FER at  $E_b/N_0 = 3.5$  dB. With  $T = 32$ , we can reap all the benefits of the oracle-assisted SC decoder, since the  $T = 32$  curve is shifted to the right with respect

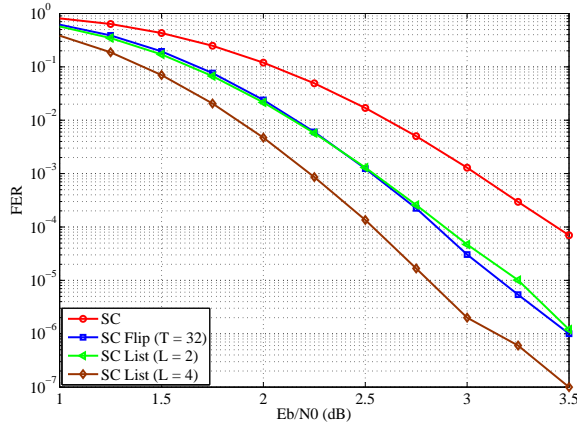


Fig. 8: Frame error rate of SC decoding, SC flip decoding with  $T = 32$  and SC list decoding with  $L = 2, 4$  for a polar code of length  $N = 1024$  and  $R = 0.5$ .

to the oracle-assisted curve by an amount that corresponds exactly to the rate loss incurred by the 16-bit CRC.

In Fig. 7 we depict the same curves for a codeword length  $N = 4096$ , while keeping the ratio  $\frac{T}{N}$  constant. We observe that it seems to become more difficult to reach the bound performance of the oracle-assisted SC decoder. As  $N$  increases, the channels get more polarized, which would suggest the opposite behavior. However, at the same time, the absolute number of the possible positions for the first error increases as well. Our results suggest that the aforementioned negative effect negates the positive effect of channel polarization.

In Fig. 8, we compare the performance of standard SC decoding, SC flip decoding, and SC list decoding. We observe that the performance of the SC flip decoder with  $T = 32$  is almost identical to that of the SC list decoder with  $L = 2$ , but with half the computational complexity at high  $E_b/N_0$  values and half the memory complexity at all  $E_b/N_0$  values. For higher list sizes, such as  $L = 4$ , SC list decoding outperforms SC flip decoding, at the cost of significantly higher complexity, since the performance of SC flip decoding is limited by the fact that it can only correct a single error.

#### D. Average Computational Complexity of SC Flip Decoding

In Fig. 9, we compare the average computational complexity of standard SC decoding, SC list decoding, and SC flip decoding. We observe that, as predicted by Proposition 2, at low SNR the average computational complexity of SC flip decoding is  $(T + 1)$  times larger than that of SC decoding but at higher SNR the computational complexity is practically identical to that of SC decoding. Moreover, the energy-proportional behavior of SC flip decoding is evident since, contrary to SC list decoding, the computational complexity decreases rapidly with decreasing difficulty of the decoding problem (i.e., increasing SNR). We also emphasize that SC flip decoding is not a viable option for the low SNR region, but this is not a region of interest for practical systems because the FER is very high.

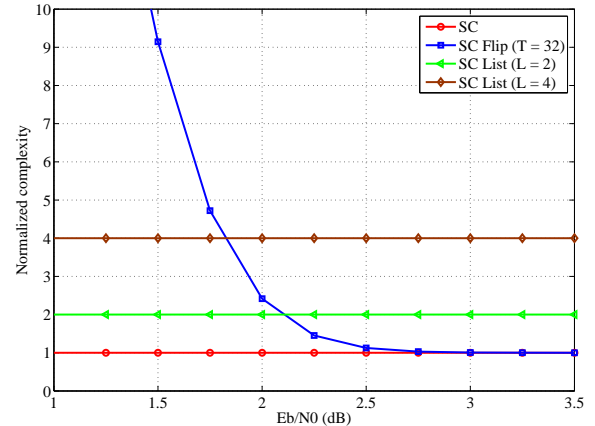


Fig. 9: Average complexity of SC flip decoding normalized with respect to the complexity of SC decoding for a polar code of length  $N = 1024$  and  $R = 0.5$ .

## V. CONCLUSION

In this paper we have introduced *successive cancellation flip* decoding for polar codes. This algorithm improves the frame error rate performance by opportunistically retrying alternative decisions for bits that turned out to be unreliable in a failing initial decoding iteration. By exploring alternative passes in the decoding tree one after another until a correct codeword is found, the average complexity and memory requirements are kept low, while approaching the performance of more complex tree-search based decoders.

## REFERENCES

- [1] E. Arkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.
- [2] C. Leroux, I. Tal, A. Vardy, W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," *Proc. IEEE Int. Conf. Acoustics, Speech and Sig. Proc.*, pp. 1665–1668, May 2011.
- [3] I. Tal, A. Vardy, "List decoding of polar codes," *Proc. IEEE Int. Symp. Inf. Theory*, pp. 1–5, Jul. 2011.
- [4] K. Chen, K. Niu, J. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Trans. Commun.*, vol. 61, no. 8, pp. 3100–3107, Aug. 2013.
- [5] B. Li, H. Shen, D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Comm. Letters*, vol. 16, no. 12, pp. 2044–2047, Dec. 2012.
- [6] G. Sarkis, P. Giard, A. Vardy, C. Thibault, W. J. Gross, "Increasing the speed of polar list decoders," arXiv:1407.2921, Jun. 2014.
- [7] C. Cao, Z. Fei, J. Yuan, J. Kuang, "Low complexity list successive cancellation decoding of polar codes," arXiv:1309.3173, Sep. 2013.
- [8] R. Pedarsani, S. Hassani, I. Tal, and E. Telatar, "On the construction of polar codes," in *Proc. IEEE Int. Symp. Inf. Theory*, pp. 11–15, Aug. 2011.
- [9] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Jul. 2013.
- [10] G. Sarkis and W. J. Gross, "Polar codes for data storage applications," in *Proc. Int. Conf. on Comp., Netw. and Comm. (ICNC)*, pp. 840–844, Jan. 2013.